AMENDMENTS TO THE SPECIFICATION

Please replace Paragraph **[0028]** with the following amended paragraph:

[0028] The `eventDispatcher` class ~~52~~ 42 represented in Figure 4 dispatches a list of events to an interested listener. This class is abstract and provides a mechanism for dispatching and filtering out all but the relevant events of interest, utilizing the `isRelevant` 52 method. By default, all methods are of interest, but classes that extend `eventDispatcher` class 42 can change this default. The `eventDispatcher` class 42 does not specify how or when events are eventually dispatched, or the type of events to be dispatched. However, events are dispatched in the thread group for which the first event is received, or within the GUI thread. Class `eventDispatcher` ~~52~~ 42 includes the following variables:

```
private static final String NAME = "EventDispatcherThread";
private List events;
private ThreadGroup group;
private boolean dispatchInSwing;
private String threadName.
```

Please replace Paragraph **[0035]** with the following amended paragraph:

[0035] Referring now to Figure 5, class `delayedEventDispatcher` 44 extends `eventDispatcher` class ~~52~~ 42 shown in Figure 4. Class `delayedEventDispatcher` 44 dispatches a list of relevant events after a specified delay has passed without any further events since the most recently received event. To avoid cases in which the list of relevant events is never dispatched because the specified delay is never exceeded, events are always delivered after a specified maximum length of time has elapsed since the most recently received event. Class `DelayedEventDispatcher` includes the following variables:

```
private long delay;
private long maxDelay;
private long lastEventTimeStamp;
private long maxDelayTimeStamp;
private boolean scheduled;
private RestartableTimer timer.
```

Please replace Paragraph **[0051]** with the following amended paragraph:

[0051] Referring now to Figure 6, class `delayedModelEventDispatcher` 46 is a model listener that dispatches a list of changes to the model to an interested listener. The class is abstract and provides methods for listening to model events and filtering out all but relevant events of interest. Method `isRelevant` 52 (shown in Figure 4) is used to determine whether an event is relevant, but relevancy may instead be determined by classes that extend `ModelEventDispatchListener`. The default `isRelevant` method always returns "true," but classes that extend `eventDispatcher` may perform filtering by overwriting this method. In one configuration, an event is examined in method `isRelevant` and only if the event is deemed relevant will the timer be started (or reset if the received event is not the first event). There are no member variables in class `delayedModelEventDispatcher` 46. Class `delayedModelEventDispatcher` 46 is an example of a specific type of listener (in this case, a listener for model event changes for a database abstraction layer used in a storage area manager product). However, those skilled in the art, upon studying the example presented here, would be able to recognize the modifications needed for other configurations of event dispatching systems that use an interface for defining its listeners and an event object to represent an event that has occurred. Methods in `DelayedModelEventDispatcher` 46, except for constructors

108 and 110, are methods used by the listening interface being implemented. In one configuration, the type of events are "model event change." In addition, the methods are invoked upon receipt of an event, where the event corresponds to some change in the model, such as a table in the database being updated, a row being deleted, etc.

Please replace Paragraph **[0052]** with the following amended paragraph:

[0052] Classes extending `eventDispatcher` class ~~50~~ 42 can override the `isRelevant` method. The `EventDispatcher` method 54 handles received events by checking whether they are relevant and adding relevant events to a list. In one configuration, the ~~`EventDispatcher`~~ `eventDispatcher` class 42 does not have any methods to determine when received events should be dispatched, or how events should be received. Class `delayedModelEventDispatcher` 46, in one configuration, is a specific type of listener that conforms to an interface for receiving for receiving events, and that extends class `delayedEventDispatcher` 44. Class `delayedModelEventDispatcher` 46 is an example of how `delayedEventDispatcher` 44 can be used in listening for a particular type of event, by providing the ability to dispatch, a determination of when to dispatch, and a method by which events are received. Although the implementation of `delayedModelEventDispatcher` 46 is specific to a particular type of listening methodology, those skilled in the art will be able to analyze its implementation to determine those modifications that may be required for other types of listening methodologies.

Please replace Paragraph **[0061]** with the following amended paragraph:

[0061] As indicated above, abstract class `delayedModelEventDispatcher` is an example of a specific type of listener. All that is required for functionality is the actual "action"

that must be performed. One example of such an action is represented as a method in Figure 7. Class `GUIRefreshListener` 50 extends class `delayedModelEventDispatcher` 46 (shown in Figure ~~46~~ 6) and provides a method `GUIRefreshListener` 124 that creates a new `GUIRefreshListener` that requires a 5 second delay before refreshing a GUI (graphical user interface) panel, up to a maximum delay of 60 seconds. Method `dispatchEvents` 126 is the method that actually causes a GUI panel to refresh. Method `dispatchEvents` 126 takes an entire list of events and causes a GUI update after a 5 second delay between any two received events, or after a delay of 60 seconds, if an event occurs and no subsequent event occurs for 60 seconds or if an event occurs and no subsequent delay as long as 5 seconds occurs for the next 60 seconds. Upon the occurrence of the next event after a dispatch of events for a refresh has occurred, the process is repeated.

Please replace Paragraph **[0063]** with the following amended paragraph:

[0063] Referring to Figure 9, when the event dispatcher enters 146 the waiting state (such as by being changed to the waiting state at block 142 of Figure 8), it waits 148 for a period of time in one configuration before checking 150 whether the predetermined minimum time has elapsed since the most recently received relevant event. (Waiting 148 is desirable in some multithreaded embodiments, for example, to prevent excessive processor time from being wasted in a timing loop.) The predetermined first, or minimum, time limit in this configuration corresponds to the reset time at block 140 of Figure 8. If the minimum time limit has not elapsed, a further check 152 is performed to determine whether a second, or "maximum," predetermined time limit has elapsed since the ~~most recently received relevant~~ receipt of the event signal corresponding to the first change event in the list of change events (see 144). If the maximum time limit has also not elapsed, the dispatcher waits 148 again before checking the timers again. Otherwise, if either the

predetermined minimum or the predetermined maximum times have elapsed, an action described by the list of change events is performed 154 (i.e., the list of change events is "dispatched") and the list of change events is cleared. In one configuration, after the list of change events is cleared, the process of iteratively receiving event signals and adding change events, checking time limits, and dispatching lists of change events is repeated, each time with a new list of change events constructed from newly received signals.

<The remainder of page left blank intentionally>